WPI

TI - Higher order synthesis method for architectural design of LSI - involves creating new control flow graph and data dependence graph automatically using computer without waiting for completion of previous loop

AB - J08044773 The method involves using a computer to prepare a control flow graph and a data dependence graph. The output variables in a particular loop which are alike and which affect a following loop, are specified. Using the specified values, a new control flow graph and data dependence graph are created automatically without waiting for the completion of the previous loop.
- ADVANTAGE - Obtains LSI with quick processing speed. Obtains highly efficient pipeline structure by using control flow graph and data dependence graph.
- (Dwg.3/20)

PN - JP8044773 A 19960216 DW199617 G06F17/50 024pp
PR - JP19940174368 19940726
PA - (MITQ) MITSUBISHI ELECTRIC CORP
MC - T01-J15
DC - T01
IC - G06F17/50
AN - 1996-164325 [17]

PAJ

TI - AUTOMATIC HIGH-LEVEL COMPOSING METHOD

AB - PURPOSE:To provide the high-level composing method which automatically composes the architexture of a high-performance LSI with pipeline structure.
- CONSTITUTION:In a loop description, an output variable which affects a next loop is specified by the arithmetic of the loop (processes S12 and S13) and arithmetic related to the specified output variable is specified (process S14 and S15). After arithmetic is classified into two groups (processes S16 and S17) according to the data dependency relation between the loops, a control flow graph and a data dependency graph are newly generated again. The newly generated control flow graph and data dependency graph are used to compose the architexture having the pipeline structure.

PN - JP8044773 A 19960216
PD - 1996-02-16
ABD - 19960628
ABV - 199606
AP - JP19940174368 19940726
PA - MITSUBISHI ELECTRIC CORP
IN - HIGASHIDA MOTOKI
I - G06F17/50

(19)日本国特許庁（ＪＰ）　(12)公開特許公報(A)　(11)特許出願公開番号
特開平8－44773
(43)公開日　平成８年(1996)２月16日

(51)Int.Cl.⁶	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　17/50		9191－5H	Ｇ０６Ｆ　15/ 60	３６０　Ａ

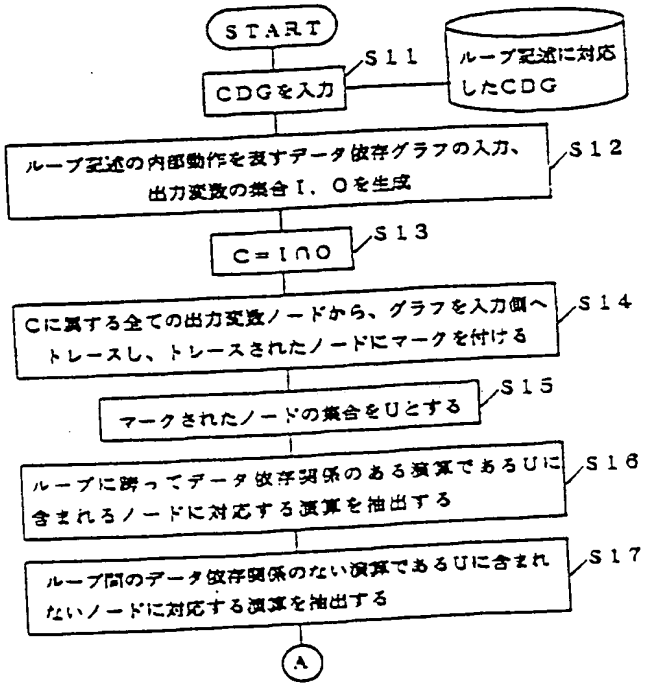
審査請求　未請求　請求項の数11　ＯＬ　（全 24 頁）

(21)出願番号　特願平6－174368
(22)出願日　平成６年(1994)７月26日

(71)出願人　000006013
三菱電機株式会社
東京都千代田区丸の内二丁目２番３号
(72)発明者　東田　基樹
兵庫県伊丹市瑞原４丁目１番地　三菱電機株式会社システムエル・エス・アイ開発研究所内
(74)代理人　弁理士　吉田　茂明　（外２名）

(54)【発明の名称】　自動高位合成方法

(57)【要約】
【目的】　パイプライン構造を有する高性能なＬＳＩのアーキテクチャを自動合成する高位合成方法を得ることを目的とする。
【構成】　ループ記述のうち、そのループでの演算によって次のループに影響を与える出力変数を特定し（工程Ｓ１２，Ｓ１３）、特定された出力変数に関わる演算を特定する（工程Ｓ１４，Ｓ１５）。そして、ループ間のデータ依存関係によって、２つのグループに分類した後（工程Ｓ１６，Ｓ１７）、新たに、コントロールフローグラフとデータ依存グラフとを再度生成する。
【効果】　新たに生成されたコントロールフローグラフとデータ依存グラフを用いることによって、パイプライン構造を持ったアーキテクチャの合成が可能になる。



1

【特許請求の範囲】

【請求項１】　ループ記述に対応したコントロールフローグラフ及びデータ依存グラフを準備する工程と、
計算機を用いて、自動的に、前記ループ記述に対応したコントロールフローグラフ及び前記データ依存グラフを、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフへと変形する変形工程とを備える、自動高位合成方法。

【請求項２】　前記変形工程は、
前記ループ記述の内部動作を表す前記データ依存グラフの入力変数を要素とする第１の集合と、前記データ依存グラフの出力変数を要素とする第２の集合とを生成する工程と、
前記第１の集合と前記第２の集合との積集合を生成する工程と、
前記積集合に属する出力変数を出力するノードから、前記データ依存グラフを入力側へトレースして、前記出力変数を出力するために必要な演算を行う第１のノードとそれ以外の演算を行う第２のノードとに分離抽出する工程と、
前記新たなデータ依存グラフとして、前記第１のノードからなる第１のデータ依存グラフと前記第２のノードからなる第２のデータ依存グラフとを生成するとともに、前記新たなコントロールフローグラフとして、前記第１及び第２のデータ依存グラフの実行手順を規定するコントロールフローグラフを生成する工程とを含む、請求項１記載の自動高位合成方法。

【請求項３】　前記計算機に準備した演算器のデータの中から、前記新たなデータ依存グラフ中の全ての前記ノードにそれぞれ一対一に対応する演算器のデータを関連づけるバインディング工程と、
前記計算機において、前記演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための前記演算器の制御論理及びデータパス構造を自動生成する生成工程とをさらに備える、請求項１または請求項２記載の自動高位合成方法。

【請求項４】　前記生成工程は、
前記ループの脱出条件を判定するノードを第１のステップに割り付ける第１のスケジューリング工程と、
前記第１のノードに対応する前記演算器について、前記第１のデータ依存グラフの制約に基づいて、第２のステップ以降に順に割り付け、制御ステップの決定を行う第２のスケジューリング工程と、
前記第２のノードに対応する前記演算器について、前記第２のデータ依存グラフの制約に基づいて、制御ステップの決定を行う第３のスケジューリング工程と、
接続すべき前記演算器の出力と入力の前記制御ステップが異なっているときには、その差異段数と等しい数のパイプラインレジスタを挿入した後に、差異段数がないときにはそのまま通常のリソース接続を行うリソース接続工程とを備える、請求項３記載の自動高位合成方法。

2

【請求項５】　前記演算器の遅延値が１クロック周期以上の場合に、前記演算器をラッチで分離されたパイプライン演算器としてモデル化する工程をさらに備える、請求項３または請求項４記載の自動高位合成方法。

【請求項６】　前記生成工程は、遅延値が１クロック周期以上の前記演算器に対応して一つの前記ループの処理を完了するために必要な工程数であるパイプラインピッチを増やす工程を含むことを特徴とする、請求項３ないし請求項５のいずれかに記載の自動高位合成方法。

【請求項７】　前記生成工程は、一つの前記ループの処理を完了するために必要な工程数であるパイプラインピッチを増やすとともに複数の前記ノードにおいて一部の演算器を共有させる工程を含むことを特徴とする、請求項３ないし請求項６のいずれかに記載の自動高位合成方法。

【請求項８】　前記新たなコントロールフローグラフ及びデータ依存グラフによって表される回路の一部のパイプラインレジスタの占有面積と前記パイプラインレジスタによって与えられる値を出力する所定の演算を行う回路の占有面積とを比較する比較工程と、
前記比較工程によって、前記所定の演算を行う回路を用いる方が回路面積を小さくできると判断された場合に、前記パイプラインレジスタを前記所定の演算を行う回路に置換する置換工程とをさらに備える、請求項３ないし請求項７のいずれかに記載の自動高位合成方法。

【請求項９】　前記ループ記述に対応したコントロールフローグラフ及びデータ依存グラフを準備する工程は、
前記コントロールフローグラフ及びデータ依存グラフで表すべき回路を表現した動作レベルのハードウェア記述言語による記述を入力する工程を含み、
前記変形工程で変形された前記新たなコントロールフローグラフ及びデータ依存グラフを表現した、論理合成ツールが直接処理可能なレジスタトランスファレベルのハードウェア記述言語による記述を出力する工程をさらに備える、請求項１ないし請求項８のいずれかに記載の自動高位合成方法。

【請求項１０】　複数の演算によって構成される一つの関数を表す新たな演算器を追加登録する工程と、
前記新たな演算器を他の前記演算器と同様に処理するために、前記動作レベルのハードウェア記述言語で特殊なコメントを挿入する工程と、をさらに備える、請求項９記載の自動高位合成方法。

【請求項１１】　前記ループ記述で表現されたループのうち複数回分のループを、１回分のループとして表現するループ記述に自動展開する工程をさらに備える、請求項９記載の自動高位合成方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】この発明は、ＬＳＩのアーキテクチャ設計に用いられる高位合成方法に関し、特にループを含んだ記述から、パイプライン構造をもった高性能なＬＳＩのアーキテクチャを自動合成する高位合成方法に関するものである。

【０００２】

【従来の技術】図１６は、従来の高位合成方法の手順を示すフローチャートである。数１に示す１つのＦＯＲループを含んだ入力記述を例にして、図１６に沿って従来の高位合成方法を説明する。

【０００３】

【数１】

```
for(i=0;i<N;i=i+1)begin
  Adr=i+base;       /*set address*/
  Data=Input(Adr); /*input from a port*/
  Res=Data*Coef;    /*calculation*/
  output(Adr,Res); /*output to a port*/
end
```

【０００４】はじめに、工程Ｓ３１において、準備された動作レベルのハードウェア記述言語による入力記述をアセンブラコードのようなプリミティブな演算の系列へと変換する。続いて、工程Ｓ３２において、その系列から、コントロールフローグラフとデータ依存グラフを生成する。ソフトウェアでは、通常、先に記述されたステートメントから逐次実行される。しかし、ハードウェアによる実行では、これらのステートメントを可能ならば、並列に実行することをコンピュータが考慮する。コントロールフローグラフとデータ依存グラフは、この並列実行の可能性を表現している。

【０００５】図１７は、工程Ｓ３２において生成された、数１の記述に対応するコントロールフローグラフとデータ依存グラフである。図１７において、１は数１のループ記述に対応するループ変数の初期化（ｉ＝０）を行うブロック、２は数１のループ記述に対応する脱出条件の比較（ｉ＜Ｎ）を行うブロック、３は数１のループの内部動作及びカウンタのインクリメント（ｉ＝ｉ＋１）を行うブロックである。このように数１のループ記述は、３つのブロック１～３に分割できる。

＊【０００６】数１に示したＦＯＲループに対応した処理を実行するためには、ブロック１を始めに実行し、続いてブロック２、ブロック３を交互に実行する必要がある。この実行順を破線のコントロールフローグラフが表現している。一方、各ブロックの内部動作については、変数の依存関係が各演算の実行順序に制約を与える。この制約を表したものが、データ依存グラフである。例えば、output（Adr,Res）という演算がノードＥ６で行なわれる為には、２つの変数Adr,Resの値が必要である。この時、変数Adrを求めるための演算（Adr＝ｉ＋base）を行う加算演算のノードＥ３と、変数Resを求めるための演算（Res＝Date*Coef）を行う乗算演算のノードＥ５から、実線の枝がひかれる。枝の存在は、それらの演算にデータ依存関係があることを意味し、それらを並列（同時）に動作させることはできない。

【０００７】一方、ノードＥ２，Ｅ３の加算演算（ｉ＝ｉ＋１，Adr＝ｉ＋base）は、互いに依存関係がないので、並列に実行可能である。高位合成方法では、はじめに、入力記述をこのようなコントロールフローグラフとデータ依存グラフへ表現し直し、以降の高位合成の処理は、これらのグラフを基にして進める。以後、コントロールフローグラフとデータ依存グラフをＣＤＧと呼ぶ。

【０００８】続いて、工程Ｓ３５において、ＣＤＧに基づいてリソースバインディング、スケジューリング、リソース接続、制御論理の生成が行なわれる。リソースバインディングでは、データ依存グラフ上の各ノード（演算）を実行する演算器（リソース）を決定する。スケジューリングでは、ＣＤＧにより規制された演算の実行順の範囲内で、必要な制御ステップ数やレジスタ数や接続の複雑度等を考慮して、演算器を動作させるべき制御ステップを決定する。リソース接続では、必要に応じて、セレクタやレジスタを挿入しながら、リソース間の接続を行なう。制御論理の生成では、スケジューリング結果を状態遷移として表現するとともに、適当な状態（制御ステップ）で、演算器やレジスタ、セレクタに制御信号を出力するような制御論理を生成する。

【０００９】

＊【表１】

ノード	演算	バインドした演算器の種類	演算器名
Ｅ１	＜	ＬＳＳ	ＬＳＳ
Ｅ２	＋	ＡＤＤ１	ＡＤＤ１－１
Ｅ３	＋	ＡＤＤ１	ＡＤＤ１－２
Ｅ４	input	ＩＮＰ	ＩＮＰ
Ｅ５	＊	ＭＬＰ１	ＭＬＰ１
Ｅ６	output	ＯＵＴ１	ＯＵＴ

5　　6

【００１０】表１に、リソースバインディングの一例を示す．ここでは、各ノードに、１つの異なった演算器を割り付けた．例えば、ノードＥ１にはデータの大小を比＊較する演算を行う演算器ＬＳＳを割り付けた．

【００１１】

【表２】

演算器名	ＬＳＳ	ＡＤＤ１-１	ＡＤＤ１-２	ＩＮＰ	ＭＬＰ１	ＯＵＴ
ステップ０	Ｅ１					
ステップ１		Ｅ２	Ｅ３			
ステップ２				Ｅ４		
ステップ３					Ｅ５	
ステップ４						Ｅ６

【００１２】また、表１のリソースバインディングに対する、スケジューリングの一例を表２に示す．このスケジューリングでは，各演算器の遅延時間は１クロック周期以内であると仮定している．図１８は、このようなリソースバインディングとスケジューリングに対してリソ※ース接続を行なった結果得られたデータパス構造を示すブロック図である．

【００１３】

【表３】

現状態	演算器名			レジスタ				次状態
	ＩＮＰ	ＯＵＴ	セレクタ	ｉ	Ａｄｒ	Ｄａｔａ	Ｒｅｓ	
ステート０			select 2	WRITE				ステート０
ステート１								ステート２（Ｓ１＝１） ステート６（Ｓ１＝０）
ステート２			select 1	WRITE	WRITE			ステート３
ステート３	active					WRITE		ステート４
ステート４							WRITE	ステート５
ステート５		active						ステート１
ステート６								ステート６

【００１４】また、表３が、制御論理生成の結果である．図１８は、ＬＳＩのデータパス構造を、表３はＬＳＩを制御するための状態遷移を表現している．図１８、及び表３により、数１の入力記述を実現するＬＳＩのアーキテクチャが完成している．以上の処理で、ＬＳＩのアーキテクチャの合成ができる．

【００１５】最後に、工程Ｓ３６において、合成されたアーキテクチャを論理合成ツールに直接入力可能なレジスタ・トランスファ・レベル（ＲＴＬ）のハードウェア記述言語による記述として出力して、高位合成が終了する．

【００１６】

【発明が解決しようとする課題】以上のように、従来の高位合成方法で合成された図１８及び表３のアーキテクチャでは、数１の入力記述のループがＮ回まわるにも関わらず、１回のループを５ステップで実行し、全動作が★（５×Ｎ＋１）ステップで終了する．従来の高位合成方法では、これ以上、高性能なアーキテクチャ（５×Ｎステップ以下のステップ数で処理できるアーキテクチャ）を合成することはできないという問題点があった．

【００１７】しかし、数１の動作記述を実現する、より高性能なアーキテクチャは存在する．図１９に、動作終了までの、図１８のデータパス構造の各演算器の動作状況を示す．図１９において、９２で示した列はステップを表しており、９３で示した行は演算器の名称を表している．これを見ると分かるように、各演算器は、５ステップに１度しか動作せず、残りのステップでは動作していないことが分かる．例えば、演算器ＬＳＳは、ステップ０でｉ＝０の処理をし、ステップ５でｉ＝１の処理をし、ステップ１０でｉ＝２の処理をするなど、５ステップ毎に処理が行っている．この空きをなくして異なったループの動作をオーバラップさせて、パイプライン状に

実行させることにより、全動作をN＋4ステップで実行できるアーキテクチャが存在する。このようなアーキテクチャをパイプラインアーキテクチャと呼ぶ。

【0018】従来の高位合成方法では、ループ動作は、図17のようなCDGに変換される。このCDGは、ループの内部動作の終了を待ってから、次のループの実行に入る必要があることを示している。

【0019】しかし、図17のようなCDGに基づいた従来の高位合成方法では、ひとつのループの内部動作の終了を待たずに、次のループの内部動作を実行するようなパイプラインアーキテクチャを合成することはできない。

【0020】但し、従来から行われている他の高位合成方法によれば、入力記述からループをなくすことにより、パイプラインアーキテクチャを合成することができる。そして、このような高位合成方法が、"Proceedings of IEEE ISCAS 87"（pp. 382-385）においてループワインディング法として提案されている。この高位合成方法は、ループ回数Nが固定（定数）の場合に適用可能である。その手法は、ループの内部動作及びカウンタのインクリメントの動作をN回コピーし（ループの展開）、数2のようなループのない記述に変換し、この記述に対して、従来の高位合成を行なう手法である。

【0021】

【数2】


```
i=0;
Adr=i+base;       /*set address*/
Data=input(Adr); /*input from a port*/
Res=Data*Coef;    /*calculation*/
output(Adr,Res); /*output to a port*/

i=1;
Adr=i+base;       /*set address*/
Data=input(Adr); /*input from a port*/
Res=Data*Coef;    /*calculation*/
output(Adr,Res); /*output to a port*/

i=2;
Adr=i+base;       /*set address*/
Data=input(Adr); /*input from a port*/
Res=Data*Coef;    /*calculation*/
output(Adr,Res); /*output to a port*/
    :
    :
    :
i=N-1;
Adr=i+base;       /*set address*/
Data=input(Adr); /*input from a port*/
Res=Data*Coef;    /*calculation*/
output(Adr,Res); /*output to a port*/
```

【0022】数2の記述に対応するCDGは、図20に示したCDGである。数2の記述は、条件判断のない演算のみの記述であり、コントロールフローグラフは不要となる。図20のデータ依存グラフに基づけば、従来の高位合成方法によっても、パイプライン状の実行を行なうスケジューリングが可能となる。しかし、コピーを行うことでループを展開しなければならないため、ループ回数が定数である場合以外適用できない。

【0023】このように、従来の高位合成方法では、任意の回数繰り返すループを含んだ入力記述から高性能なパイプラインアーキテクチャを合成することができない。そのため、非パイプラインアーキテクチャを合成するか、人手でパイプラインアーキテクチャを用いたRTL記述を作成しなければならない。しかし、前者では高性能化が達成できず、後者では設計期間の長期化、設計者の負担の増大などの問題が生ずる。

【0024】この発明は上記のような問題を解決するためになされたもので、任意の回数繰り返すループを含んだ入力記述から、高性能なパイプライン型のデータパス構成を有するアーキテクチャ及びその制御論理を自動合成することを目的とする。

【0025】

【課題を解決するための手段】第1の発明に係る自動高

位合成方法は、ループ記述に対応したコントロールフローグラフ及びデータ依存グラフを準備する工程と、計算機を用いて、自動的に、前記ループ記述に対応したコントロールフローグラフ及び前記データ依存グラフを、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフへと変形する変形工程とを備えて構成される。

【００２６】第２の発明に係る自動高位合成方法は、第１の発明の自動高位合成方法において、前記変形工程は、前記ループ記述の内部動作を表す前記データ依存グラフの入力変数を要素とする第１の集合と、前記データ依存グラフの出力変数を要素とする第２の集合とを生成する工程と、前記第１の集合と前記第２の集合との積集合を生成する工程と、前記積集合に属する出力変数を出力するノードから、前記データ依存グラフを入力側へトレースして、前記出力変数を出力するために必要な演算を行う第１のノードとそれ以外の演算を行う第２のノードとに分離抽出する工程と、前記新たなデータ依存グラフとして、前記第１のノードからなる第１のデータ依存グラフと前記第２のノードからなる第２のデータ依存グラフとを生成するとともに、前記新たなコントロールフローグラフとして、前記第１及び第２のデータ依存グラフの実行手順を規定するコントロールフローグラフを生成する工程とを含むことを特徴とする。

【００２７】第３の発明に係る自動高位合成方法は、第１または第２の発明の自動高位合成方法において、前記計算機に準備した演算器のデータの中から、前記新たなデータ依存グラフ中の全ての前記ノードにそれぞれ一対一に対応する演算器のデータを関連づけるバインディング工程と、前記計算機において、演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための前記演算器の制御論理及びデータパス構造を自動生成する生成工程とを備えて構成される。

【００２８】第４の発明に係る自動高位合成方法は、第３の発明の自動高位合成方法において、前記生成工程は、前記ループの脱出条件を判定するノードを第１のステップに割り付ける第１のスケジューリング工程と、前記第１のノードに対応する前記演算器について、前記第１のデータ依存グラフの制約に基づいて、第２のステップ以降に順に割り付け、制御ステップの決定を行う第２のスケジューリング工程と、前記第２のノードに対応する前記演算器について、前記第２のデータ依存グラフの制約に基づいて、制御ステップの決定を行う第３のスケジューリング工程と、接続すべき前記演算器の出力と入力の前記制御ステップが異なっているときには、その差異段数と等しい数のパイプラインレジスタを挿入した後に、差異段数がないときにはそのまま通常のリソース接続を行うリソース接続工程とを備えて構成される。

【００２９】第５の発明に係る自動高位合成方法は、第３または第４の発明の自動高位合成方法において、前記演算器の遅延値が１クロック周期以上の場合に、前記演算器をラッチで分離されたパイプライン演算器としてモデル化する工程を備えて構成される。

【００３０】第６の発明に係る自動高位合成方法は、第３乃至第５の発明の自動高位合成方法において、前記生成工程は、遅延値が１クロック周期以上の前記演算器に対応して一つの前記ループの処理を完了するために必要な工程数であるパイプラインピッチを増やす工程を含むことを特徴とする。

【００３１】第７の発明に係る自動高位合成方法は、第３乃至第６の発明の自動高位合成方法のいずれかにおいて、前記生成工程は、一つの前記ループの処理を完了するために必要な工程数であるパイプラインピッチを増やすとともに複数の前記ノードにおいて一部の演算器を共有させる工程を含むことを特徴とする。

【００３２】第８の発明に係る自動高位合成方法は、第３乃至第７の発明の自動高位合成方法のいずれかにおいて、前記新たなコントロールフローグラフ及びデータ依存グラフによって表される回路の一部のパイプラインレジスタの占有面積と前記パイプラインレジスタによって与えられる値を出力する所定の演算を行う回路の占有面積とを比較する比較工程と、前記比較工程によって、前記所定の演算を行う回路を用いる方が回路面積を小さくできると判断された場合に、前記パイプラインレジスタを前記所定の演算を行う回路に置換する置換工程とを備えて構成される。

【００３３】第９の発明に係る自動高位合成方法は、第１乃至第８の発明の自動高位合成方法のいずれかにおいて、前記ループ記述に対応したコントロールフローグラフ及びデータ依存グラフを準備する工程は、前記コントロールフローグラフ及びデータ依存グラフで表すべき回路を表現した動作レベルのハードウェア記述言語による記述を入力する工程を含み、前記変形工程で変形された前記新たなコントロールフローグラフ及びデータ依存グラフを表現した、論理合成ツールが直接処理可能なレジスタトランスファレベルのハードウェア記述言語による記述を出力する工程を備えて構成される。

【００３４】第１０の発明に係る自動高位合成方法は、第９の発明の自動高位合成方法において、複数の演算によって構成される一つの関数を表す新たな演算器を追加登録する工程と、前記新たな演算器を他の前記演算器と同様に処理するために、前記動作レベルのハードウェア記述言語で特殊なコメントを挿入する工程とを備えて構成される。

【００３５】第１１の発明に係る自動高位合成方法は、第９の発明の自動高位合成方法において、前記ループ記述で表現されたループのうち複数回分のループを、１回分のループとして表現するループ記述に自動展開する工

程を備えて構成される。

【0036】

【作用】第1の発明の自動高位合成方法におけるループ記述を特定する工程によって、ループ記述に対応するコントロールフローグラフ及びデータ依存グラフを準備する。そして、変形工程において、ループ記述に対応するコントロールフローグラフ及びデータ依存グラフを新たなコントロールフローグラフ及びデータ依存グラフへと変形する。変形工程で得られた新たなコントロールフローグラフ及びデータ依存グラフは、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す。

【0037】第2の発明の自動高位合成方法における入出力変数をそれぞれ構成要素とする第1及び第2の集合を生成する工程と、それらの積集合を生成する工程とによって、前のループの演算結果を次のループの演算結果に反映するための変数を特定することができる。そして、積集合に属する出力変数を出力するノードから、データ依存グラフを入力側へトレースして、出力変数を出力するために必要な演算を行う第1のノードとそれ以外の演算を行う第2のノードとに分離抽出する工程と、新たなデータ依存グラフとして、第1のノードからなる第1のデータ依存グラフと第2のノードからなる第2のデータ依存グラフとを生成するとともに、新たなコントロールフローグラフとして、第1及び第2のデータ依存グラフの実行手順を規定するコントロールフローグラフを生成する工程とによって、パイプライン状に演算を実行可能な第2のデータ依存グラフを生成することができる。

【0038】第3の発明の自動高位合成方法におけるバインディング工程によって新たなデータ依存グラフ中の全てのノードにそれぞれ一対一に対応する演算器のデータを関連づけることができる。そうすることによって、生成工程において、計算機において、演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器の制御論理及びデータパス構造を容易に自動生成することができる。

【0039】第4の発明の自動高位合成方法における第1ないし第3のスケジューリング工程によって、演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器のスケジューリングができ、リソース接続工程によって、接続すべき演算器の出力と入力の制御ステップが異なっているときに、その差異段数と等しい数のパイプラインレジスタを挿入するとともにリソース接続を行うことでパイプライン状に実行したときにデータが消失しないようにパイプラインレジスタを挿入することができる。

【0040】第5の発明の自動高位合成方法におけるモデル化する工程で、演算器の遅延値が1クロック周期以上の場合に、演算器をラッチで分離されたパイプライン演算器としてモデル化され、そのモデル化された演算器のラッチで分離された部分を一つの演算器と同様に扱うことによって、容易に、ループ動作をパイプライン状に実行するための演算器の制御論理及びデータパス構造を容易に自動生成することができる。

【0041】第6の発明の自動高位合成方法における生成工程は、一つのループの処理を完了するために必要な工程数であるパイプラインピッチを増やすことによって、遅延値が1クロック周期以上の演算器に対して増やしたパイプラインピッチを割り当てることができ、遅延値が1クロック周期以上の演算器を用いて、リソースが空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器の制御論理及びデータパス構造を容易に自動生成することができる。

【0042】第7の発明の自動高位合成方法における複数のノードにおいて一部の演算器を共有させる工程は、一つのループの処理を完了するために必要な工程数であるパイプラインピッチを増やすことで、リソースが空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器の制御論理及びデータパス構造を容易に自動生成することができる。

【0043】第8の発明の自動高位合成方法における比較工程によれば、新たなコントロールフローグラフ及びデータ依存グラフによって表される回路の一部のパイプラインレジスタの占有面積とパイプラインレジスタによって与えられる値を出力する所定の演算を行う回路の占有面積とを比較して、占有面積の小さい方がどちらかを判断することができる。そして、置換工程において、比較工程により所定の演算を行う回路を用いる方が回路面積を小さくできると判断された場合に、パイプラインレジスタを所定の演算を行う回路に置換することで、占有面積をより小さくすることができる。

【0044】第9の発明の自動高位合成方法におけるコントロールフローグラフ及びデータ依存グラフで表すべき回路を表現した動作レベルのハードウェア記述言語による記述を入力する工程で、動作レベルのハードウェア記述言語で記述したものを入力すると、レジスタトランスファレベルのハードウェア記述言語による記述を出力する工程で、前記変形工程で変形された前記新たなコントロールフローグラフ及びデータ依存グラフを表現した、論理合成ツールが直接処理可能なレジスタトランスファレベルのハードウェア記述言語による記述を出力することができ、そのため、入力が容易になる。

【0045】第10の発明の自動高位合成方法における新たな演算器を追加登録する工程と特殊なコメントを挿入する工程とにより、複数の演算で構成される一つの関

数を表す新たな演算器を追加登録し、動作レベルのハードウェア記述言語で特殊なコメントを挿入することができ、登録された新たな演算器を他の演算器と同様に処理することが可能になる。

【００４６】第１１の発明の自動高位合成方法における、複数回数分のループを、１回のループに自動展開し、ループの展開が終了した後に、パイプラインアーキテクチャを可能にする変形工程を行うことで、ループの回数を削減したパイプラインアーキテクチャの構成ができる。

【００４７】

【実施例】

実施例１．以下、この発明の第１実施例を図について説明する。数１の入力記述を解析すると、次のループのデータ依存グラフの演算に影響を与えるのは、カウンタのインクリメント（ｉ＝ｉ+1）だけであり、他の演算についてはループ内部で完結していることがわかる。図１は、複数のループに渡って行われる処理の概念を示す図である。図１において、１０ａから１０ｙは変数ｉの値の決定と変数ｉの値の範囲の判断とを行う操作、１１ａから１１ｘは決定された変数ｉを用いて行う内部操作のうちの非依存部である。パイプライン型のデータパス構成を形成するためには、図１のように非依存部１１ａ～１１ｘの動作をオーバーラップさせパイプライン状に実行させることが必要となる。その様子を図２に示す。図２はパイプライン状に実行させたときの各演算とその演算が行われるステップの関係を示す図である。図２において、１２はステップ回数を表示する列、１３はノードの符号を示した行である。また、図において、１４～１６は変数ｉの値が０～２の時の処理が行われることを示す記号、１７～１９は変数ｉの値がＮ－３～Ｎ－１の時の処理が行われることを示す記号である。例えば、ノードＥ１及びノードＥ２は操作１０ａ～１０ｙに属し、ノードＥ３～Ｅ６は操作１１ａ～１１ｘに属するものとすると、パイプライン状に処理を実行するノードＥ３～Ｅ６は、ノードＥ１またはノードＥ２の出力を用いた処理ができる。

【００４８】図３及び図４に、ループ記述に対応したＣＤＧから、パイプライン状の実行が可能なＣＤＧへ変換する処理のフローチャートを示す。コンピュータにおいて、図１７のＣＤＧを入力して、パイプライン状の実行が可能なＣＤＧへの変換手順をフローチャートに沿って説明する。

【００４９】まず、工程Ｓ１１において、図１７に示した従来の高位合成方法でループ記述から変換されたＣＤＧを入力する。次に、工程Ｓ１２において、ループの内部動作を表すデータ依存グラフ（図１７のブロック３）に外から入ってくる変数の集合Ｉと外へ出ていく変数の集合Ｏを生成する。図１７の例では、集合Ｉ＝｛1. i. base. Coef｝、集合Ｏ＝｛i｝　となる。

【００５０】続いて、工程Ｓ１３において、集合Ｉと集合Ｏの交差集合（積集合）Ｃを求める。図１７の例では、交差集合Ｃ＝｛i｝　となる。次に、工程Ｓ１４において、ブロック３に対して、交差集合Ｃに含まれる出力変数ｉから、入力に向かって枝をトレースする。

【００５１】工程Ｓ１５において、先の工程Ｓ１４でトレースされたノード（演算）の集合Ｕを求める。図１７の例では、集合Ｕは加算ノードＥ２となる。工程Ｓ１６において、集合Ｕの要素を、次のループに影響を与える演算として抽出する。工程Ｓ１７において、集合Ｕに含まれない演算を、ループ間に依存関係のない演算として抽出する。

【００５２】工程Ｓ１８において、次のループに影響を与える演算の集合と、依存関係のない演算の集合について新たなブロックをつくり、それぞれのブロックへ、条件比較ブロック２からコントロール枝を引く。さらに、依存関係のあるブロックについては、ループのためのコントロール・フローの枝を、そのブロックから条件比較ブロックへ引き、コントロールフローグラフを生成する。そして、それぞれのブロック内で、再度データ依存グラフを構成しなおす。工程Ｓ１９において、変形することにより得られたＣＤＧを出力する。以上のような処理により、図１７のＣＤＧを、図５のようなパイプライン状の実行が可能なＣＤＧへ変形できる。

【００５３】以上のようにして第１の実施例の高位合成方法は、ループ記述に対応するＣＤＧを、図５のようなパイプライン状の実行が可能な形態のＣＤＧへと変換することができる。図５において、４はカウンタのインクリメント（ｉ＝ｉ＋１）を行うブロック、５は数１のループ依存関係の無い内部動作を行うブロックである。

【００５４】図５のＣＤＧは、図１７のＣＤＧを、ループ間にまたがって依存関係のある演算のノードＥ２を含むブロック４と依存関係のない演算のノードＥ３～Ｅ６のみからなるブロック５に分けたものである。図５に示したＣＤＧは、非依存部の演算については、前のループの演算終了を待つことなく次のループの演算実行が可能であることを意味している。

【００５５】次に、図５に示したＣＤＧを用いて、パイプラインアーキテクチャの合成についての説明を続ける。

【００５６】図６及び図７は、パイプラインアーキテクチャ合成のための処理を行なうフローチャートである。図５のＣＤＧを入力例として、このフローチャートを具体的に説明する。

【００５７】まず、工程Ｓ２１において、図５に示したような変形されたＣＤＧをコンピュータが入力する。次に、工程Ｓ２２において、その演算を実行可能な演算器をＣＤＧの個々の演算に対して１つ割り付ける。この時、割り付けられる各演算器の最大遅延時間は、１クロック周期以内である必要がある。また、同種の演算器を

割り付ける場合、１つの演算器を共有せず、新たな演算器を用意し、それを割り付ける。この演算器の割り付け処理は、リソースバインディングと呼ばれている。

*【００５８】
【表４】

演算器の種類	最大遅延（ｎＳｅｃ）	コスト（面積）	可能な演算
ＡＤＤ１	３０	１００	＋
ＡＤＤ２	６０	５０	＋
ＭＬＰ１	４５	１０００	＊
ＭＬＰ２	９０	５００	＊
ＬＳＳ	２０	２０	＜
ＩＮＰ	４５	１０	input
ＯＵＴ１	４５	５０	output
ＯＵＴ２	９０	１０	output

【００５９】表４に示すような演算器が利用できるものとし、それらの演算器のデータがコンピュータに準備されているものする。クロック周期を50nSecとした時の、図５のＣＤＧに対するリソースバインディングの一例は表１で示される。表４を分析すると、演算器ＡＤＤ２，ＭＬＰ２，ＯＵＴ２の演算器は、遅延が１クロック周期（50nSec）より長いので使用できない。また、２つの加算に対しては、異なった演算器ＡＤＤ1-1，ＡＤＤ1-2を割り付ける。

【００６０】工程Ｓ２３～工程Ｓ２５において、これらの演算器が、どの制御ステップにて、どの演算を実行するかを決定する。この制御ステップを決定する処理は、スケジューリングと呼ばれている。

【００６１】工程Ｓ２３において、ループの脱出条件を判定する演算は、第０ステップにて実行させるように割り付ける。ループにまたがって依存関係のある全ての演算は、第１ステップにて実行させる。もし、第１ステップにて実行できなければ、高位合成不能である。つまり、第１のステップに依存関係の無いノードが割り付けられると処理ができなくなるからである。

【００６２】最後に、次のループへ影響しない演算に対して、制御ステップを決定する。この問題は、必要なレジスタ数を評価関数としたスケジューリング問題となる。スケジューリング問題の解法については、従来の高位合成と同様の手法を用いる。Daniel Gajski,Nikil Dutt,Allen Wu,Steve Lin著、"High-Level Synthesis"、（１９９２）、KLUWERACADEMIC PUBLISHERSに様々な解法が紹介されている。演算をできるだけ早いステップで動作させるスケジューリング法である、ＡＳＡＰ法と呼ばれる解法を用いて、図５のＣＤＧと表１のリソースバインディングに対して、スケジューリングした結果は表２に示したものと同様である。

※【００６３】工程Ｓ２６において、使用された演算器を直接、または、レジスタやセレクタを介して接続し、データパス構造を生成する。この処理はリソース接続と呼ばれている。リソース接続は、次のようなルールに従って行なわれる。データ依存グラフにおいて、データの授受を行う関係にある２つのノードに対応する演算器の制御ステップが前後の関係にあるなら、言い換えると、それらの演算器の入力と出力とを同じ制御ステップで行うことができるなら、演算器同士を直接接続する。もし、動作すべき制御ステップが離れていれば、制御ステップの差分の段数のレジスタ（パイプラインレジスタと呼ぶ）を介して接続する。

【００６４】また、次のループに影響を与えるオペレーションには、ループ内動作に対応する演算器の出力端子にレジスタを挿入し、次の動作の入力へ接続する。但し、同一の演算器やレジスタの入力端子に、複数の演算器やレジスタを接続する必要がある場合には、セレクタを生成し、セレクタを介して接続する。図８に、表２のスケジューリング結果に対して、リソース接続を行なった例を示す。図８において、ｒ１～ｒ１１はレジスタ、２０は第２の入力端子に接続されたレジスタｒ２に記憶されている値０と第１の入力端子に入力される値とを切り替えて出力するセレクタ、２１はセレクタ２０が出力した値を保持しているレジスタｒ４の値とレジスタｒ１が保持しているループの繰り返し回数の値とを比較して結果に応じて信号Ｓ１を出力する比較器、２２はセレクタ２０等の各演算器を制御するための制御論理回路ブロック、２３はレジスタｒ３に記憶されている値１とレジスタｒ４に記憶されている値とを加算してその結果をセレクタ２０の第１の入力端子に与える加算器、２４はレジスタｒ４に記憶されている値とレジスタｒ５に記憶されている値とを加算してその結果をレジスタｒ６に出力

17

する加算器、２５はレジスタｒ６の値に応じてデータを入力してレジスタｒ７に出力する入力装置、２６はレジスタｒ７に記憶されているデータとレジスタｒ８に記憶されている係数とを掛け合わせてその結果をレジスタｒ１１に出力する乗算器、２７はレジスタｒ１０とレジスタｒ１１との値に応じた出力を行う出力装置である。なお、レジスタｒ６からレジスタｒ９へ、またレジスタｒ９からレジスタｒ１０へステップ毎に順に記憶されているデータが送られる。以上で、パイプライン動作部のデータパス構造が合成できた。

【００６５】続いて、工程Ｓ２７において、制御論理回路ブロック２２が出力するための制御論理を生成する。制御論理は、ＬＳＩの動作を制御するための、状態遷移として表現され、適切な状態での演算器やレジスタやセレクタへの動作信号の発生や、条件に応じた次状態を規定している。制御論理の生成法を説明する。スケジューリングで求められた制御ステップの個数を（Ｎ＋１）個とする。この時、（２×Ｎ＋１）個の状態を用意する。（２×Ｎ＋１）個の状態は、(1) 初期値設定、(2) 初期判断、(3) パイプラインの導入部、(4) フル・パイプライン動作部、(5) パイプライン導出部のブロックに分割できる。

【００６６】ステート０は、初期値設定ブロックである。ここでは、カウンタに対応するレジスタに初期値を設定する。設定に必要なセレクタとレジスタへの制御信号を発生する。次状態は、ステート１とする。

【００６７】ステート１は、初期判断ブロックである。これは、ループを一度も実行しないケースを想定している。この状態においては、ループの脱出条件判定を行な＊

18

＊う演算器が動作する。この演算器に対する制御信号が必要ならその制御信号を発生する。図８の例では、不要であるため、制御信号は生成しない。次状態は、ループの条件判定の演算器の出力が真(1) ならステート２とし、偽(0) なら、終状態（ＥＮＤ）とする。

【００６８】ステート２からステートＮまでは、パイプライン導入部である。ステート２からステートＮの中の任意のステートｉでは、スケジューリング結果のステップ０～ｉ－１までの制御ステップに関わる演算器、レジスタ、セレクタに対して、必要な制御信号を生成する。次状態はループの条件判定の演算器の出力が真(1) ならステートｉ＋１とし、偽(0) なら、ステートＮ＋２とする。

【００６９】ステートＮ＋１は、フル・パイプライン動作部である。この状態では、全ての演算器を動作させる。次状態は、ループの条件判定の演算器の出力が真(1) ならステートＮ＋１（同じ状態）とし、偽(0) なら、ステートＮ＋２とする。

【００７０】ステートＮ＋２からステート２Ｎまでは、パイプライン導出部である。ステートＮ＋２からステート２Ｎまでの任意のステートｉでは、スケジューリング結果のステップｉ－Ｎ～Ｎまでの制御ステップに関わる演算器、レジスタ、セレクタに対して、必要な制御信号を生成する。次状態はステート２Ｎ以外の状態では、ステートｉ＋１とし、ステート２Ｎの状態では、終状態（ＥＮＤ）とする。

【００７１】

【表５】

現状態	演算器名			レジスタ					
	INP	OUT	セレクタ	i	Adr	Data	Adr (1)	Res	Adr (2)
ステート０			select 2	WRITE					
ステート１									
ステート２			select 1	WRITE	WRITE				
ステート３	active		select 1	WRITE	WRITE	WRITE	WRITE		
ステート４	active		select 1	WRITE	WRITE	WRITE	WRITE	WRITE	WRITE
ステート５	active	active	select 1	WRITE	WRITE	WRITE	WRITE	WRITE	WRITE
ステート６	active	active				WRITE	WRITE	WRITE	WRITE
ステート７		active						WRITE	WRITE
ステート８		active							

【００７２】

※ ※【表６】

現状態	次状態	
ステート0	ステート1	初期値設定
ステート1	ステート2（S1=1） END　（S1=0）	初期値判断
ステート2	ステート3（S1=1） ステート6（S1=0）	パイプライン導入部
ステート3	ステート4（S1=1） ステート6（S1=0）	
ステート4	ステート5（S1=1） ステート6（S1=0）	
ステート5	ステート5（S1=1） ステート6（S1=0）	フル・パイプライン
ステート6	ステート7	パイプライン導出部
ステート7	ステート8	
ステート8	END	

【００７３】表５，表６に、上記の高位合成にて得られる制御論理を示す．図８に示した構成と、表５及び表６に示した制御理論によって、数１の入力記述を実現するパイプラインアーキテクチャが得られている．

【００７４】最後に、工程Ｓ２８において、得られたアーキテクチャを出力する．以上の処理により、パイプラインアーキテクチャを合成することができる．また、リソースコストの増加を抑えて処理の高速化を実現することができる．

【００７５】実施例２．次に、この発明の第２実施例による高位合成方法について説明する．上記の高位合成方法では、各演算器の最大遅延時間が、１クロック周期以内である必要がある．しかし、このような制限があると、使用可能な演算器が限定されてしまい、与えられる条件によっては、このような制限のためにアーキテクチャの合成が不可能な場合が出てくる．

【００７６】まず、このような問題点を解消するための概念について説明する．図９に示すように、出力の入力に対して遅延時間Delayの組合せ回路のみからなる演算器３０があり、遅延時間Delay＞１クロック周期Clock-Periodとする．この演算器を用いた場合、一度データを入力すると、出力結果が得られるまでの、（int（Delay／Clock-Period）＋１）クロック周期の間、この演算器を使用することができない．但し、int（ｘ）は、実数ｘを越えない最大の整数を与える関数であり、（Delay／Clock-Period）が整数の時は例外的に（Delay／Clock-Period）クロック周期の間使用できない．パイプラインアーキテクチャでは、データをクロック毎に演算器に投入し、処理できることが必要である．したがって、このような演算器は、パイプラインアーキテクチャでは使用できない．このような演算器を使用可能とする手法に、パイプラインラッチの挿入法がある．まず、各段の実行が１クロック周期内に終るように演算器をＮ段のブロックに分割する．分割されたブロック間の接続にラッチを挿入する．パイプラインラッチの挿入の例を、図１０に示す．図１０において、組合せ回路３１～３６で組合せ回路３０と同等の機能を有している．そして、組合せ回路３１，３３の出力はそれぞれラッチ３２，３４に保持される．このようにラッチが挿入された演算器には、データをクロック毎に投入することができる．但し、演算器の計算結果が得られるのは、Ｎクロック周期後である．遅延時間が１クロック周期以上の演算器に対しては、このようにして、パイプラインラッチを挿入した演算器へ設計者が変換し、あるいは自動変換し、その後上記と同様の高位合成方法を用いて、パイプラインアーキテクチャの合成を行なう．

【００７７】実施例３．しかし、上記のような回路を分割する方法は、組合せ回路のみからなる演算器の場合に適用可能である．ところが、例えば、メモリや外部ポートとのインターフェース回路等では、演算器は組合せ回路のみで構成されず、上記の方法では、パイプラインアーキテクチャを合成することはできない．

【００７８】１クロック周期以上の遅延を持つ演算器が組合せ回路以外の回路で構成されている場合も、パイプラインアーキテクチャを合成可能とするこの発明の第３実施例による高位合成方法について説明する．

【００７９】表４の利用可能な演算器のテーブルにおいて、ＯＵＴ１という種類の演算器がなかったとする．クロック周期を50nSecとすれば、出力に対応する演算器で、遅延が１クロック周期以内のものは存在しない．したがって、２クロック周期必要な演算器ＯＵＴ２を用いて、パイプラインアーキテクチャを合成することを考える．

【００８０】

【表７】

21　　22

演算器名	LSS	ADD1-1	ADD1-2	INP	MLP1	OUT
ステップ0	E1					
ステップ1		E2	E3			
ステップ2				E4		
ステップ3					E5	
ステップ4						E6
ステップ5						E6

【0081】OUT2の動作には2クロック必要であるため、表2のスケジューリングが、表7のように変化する。図6及び図7のフローチャートに示した手順を含む高位合成方法では、各演算器は一つの制御ステップでのみ動作する必要がある。そのため、表7のようなスケジューリング結果に対しては、正しい制御論理を生成することができない。

【0082】そこで、制御論理の生成方式を以下のように変更する。スケジューリングで求められた制御ステップの個数を（N＋1）個とする。また、遅延時間が最長の演算器に必要なクロック周期をm周期とする。さらに、L＝ra（N／m）とする。ただし、ra（a）は、実数aの少数点以下を切り上げた整数を意味する。

【0083】この時、（2mL－m＋2）の状態を用意する。これらの状態は、(1) 初期値設定、(2) 初期判断、(3) パイプラインの導入部、(4) フル・パイプライン動作部、(5) パイプライン導出部のブロックに分割できる。

【0084】初期値設定と初期判断のブロックについては、図7の工程S27と同様に、ステート0は、初期値設定ブロックである。ここでは、カウンタに対応するレジスタに初期値を設定する。設定に必要なセレクタとレジスタへの制御信号を発生する。次状態は、ステート1とする。

【0085】ステート1は、初期判断ブロックである。これは、ループを一度も実行しないケースを想定している。この状態においては、ループの脱出条件判定を行なう演算器が動作する。これらの演算器に対する制御信号が必要ならその制御信号を発生する。図8の例では、不要であるため、制御信号は生成しない。次状態は、ループの条件判定の演算器の出力が真(1) ならステート2とし、偽(0) なら、終状態（END）とする。

【0086】ステート2からステートmL－m＋1までは、パイプライン導入部である。ステート2からステート*mL－m＋1の中の任意のステートi＋1（i＝1，2，…，mL－m）に対する制御論理は以下のように定まる。t＝i　mod　mとする。このとき、スケジューリング結果のステップt，t＋m，t＋2m，…，iの制御ステップに関わる演算器、レジスタ、セレクタに対して、必要な制御信号を生成する。次状態については、tが0の時とそれ以外の時で異なる。tが0の時はループの条件判定の演算器の出力が真(1) ならステートi＋1とし、偽(0)なら、ステートmL＋2とする。tが0以外の時は、ループの条件判定の演算器の出力とは無関係にステートi＋1とする。

【0087】ステートmL－m＋2からステートmL＋1は、フル・パイプライン動作部である。ステートmL－m＋2＋i（i＝0，1，…，m－1）では、s　mod m≡iとなるようなステップsの制御ステップに関わる演算器、レジスタ、セレクタに対して、必要な制御信号を生成する。次状態については、iがm－1の時は、ループの条件判定の演算器の出力が真(1)ならステートmL－m＋2とし、偽(0)なら、ステートmL＋2とする。iがm－1以外の時は、常にステートmL－m＋2＋i＋1とする。

【0088】ステートmL＋2からステート2mL－m＋1、までは、パイプライン導出部である。ステートmL＋i＋1（i＝1，2，…，mL－m）に対する制御論理は以下のように定まる。t＝i　mod　mとする。このときスケジューリング結果のステップi，i＋m，i＋2m，…，mL－m＋2＋tの制御ステップに関わる演算器、レジスタ、セレクタに対して、必要な制御信号を生成する。次状態は、ステート2mL－m＋1以外の状態では、ステートmL＋i＋2とし、ステート2mL－m＋1の状態では、終状態（END）とする。

【0089】

【表8】

23　　24

現状態	演算器名			レジスタ					
	INP	OUT	セレクタ1	i	Adr	Data	Adr(1)	Res	Adr(2)
ステート0			select 2	WRITE					
ステート1									
ステート2			select 1	WRITE	WRITE				
ステート3	active					WRITE	WRITE		
ステート4			select 1	WRITE	WRITE			WRITE	WRITE
ステート5	active	active				WRITE	WRITE		
ステート6		active	select 1	WRITE	WRITE			WRITE	WRITE
ステート7	active	active				WRITE	WRITE		
ステート8		active						WRITE	WRITE
ステート9		active							
ステート10		active							
ステート11									

【0090】　　＊　＊【表9】

現状態	次状態	
ステート0	ステート1	初期値設定
ステート1	ステート2（S1＝1） END　（S1＝0）	初期値判断
ステート2	ステート3	パイプライン導入部
ステート3	ステート4（S1＝1） ステート8（S1＝0）	
ステート4	ステート5	
ステート5	ステート6（S1＝1） ステート8（S1＝0）	
ステート6	ステート7	フル・パイプライン
ステート7	ステート6（S1＝1） ステート8（S1＝0）	
ステート8	ステート9	パイプライン導出部
ステート9	ステート10	
ステート10	ステート11	
ステート11	END	

【0091】表8、表9に表7のスケジューリングに対して、上記の手順を適用したとき得られる制御論理を示す。この例では、N＝5，m＝2，L＝3となる。従って、ステップ2～5がパイプライン導入、ステップ6～7がフル・パイプライン部、ステップ8～11がパイプ※ライン導出部となる。

【0092】上記の方法では、フル・パイプライン部において、m個の状態でもってループを構成している。このループを構成する状態数をパイプラインピッチと呼ぶ。図6、図7を用いて説明した方法では、パイプライ

ンピッチが1であったため、フル・パイプラインの状態では、スループットとしては、1クロックあたり1ループの動作が処理されていた。一方、この方法によれば、mクロックあたり1ループの動作が処理されている。従って、スループットが低下したパイプラインアーキテクチャが合成されることとなる。

【0093】実施例4．次に、この発明の第4実施例による高位合成方法について説明する。第1実施例による高位合成方法では、グラフ中の各演算器に一対一に対応する演算器を用意する必要がある。これが不可能な場合＊には適用できない。そこで、一部の演算器を共有しても、パイプラインピッチを増やすことにより、第1実施例の高位合成方法を適用可能とする方式に関するものである。

【0094】ここでは、2つの加算（E2，E3）に対して1つの加算器が割り付けることを考える。そのため、図5のCDGに対して、表10のようなリソースバインディングを行ったとする。

【0095】

【表10】

ノード	演算	バインドした演算器の種類	演算器名
E1	<	LSS	LSS
E2	+	ADD1	ADD1
E3	+	ADD1	ADD1
E4	input	INP	INP
E5	*	MLP1	MLP1
E6	output	OUT	OUT

【0096】表11に、このようなリソースバインディングに対するスケジューリングの例を示す。

※【0097】

※【表11】

演算器名	LSS	ADD1	INP	MLP1	OUT
ステップ0	E1				
ステップ1		E3			
ステップ2		E2	E4		
ステップ3				E5	
ステップ4					E6

【0098】1つの演算器を、複数の演算にて共有するために、制御ステップをずらして動作するようにスケジューリングをする。第1実施例の高位合成方法では、各演算器は一つの制御ステップでのみ動作する必要がある。このようなスケジューリング結果に対しては、正しい制御論理を生成することができない。

【0099】このような場合においても、スケジューリング方式を変更し、さらにこの発明の第3実施例による高位合成方法の制御論理の生成方式を用いることにより、パイプラインアーキテクチャが合成可能となる。

【0100】スケジューリング方式の変更点は、スケジューリングに次のような制限を設けることである。一つ★の演算器に割り付けられた演算の最大数をmとする。また、ある演算器が動作する制御ステップをS1，S2，…，Sk（k≦m）とする。このとき、Si mod m と、Sj mod mとは等しくないという制限を設けて、スケジューリングを行なう。ただし、i，j＝1，2，…，kとして、iとjは等しくないものとする。

【0101】そして、共有する加算器に入力されるデータの切り替えを行うセレクタを新たに設ける。このセレクタは、制御論理によって、ステップ1とステップ2でその切り替えを行う。

【0102】

【表12】

27　　28

現状態	演算器名				レジスタ					
	INP	OUT	セレクタ1	セレクタ2	i	Adr	Data	Adr(1)	Res	Adr(2)
ステート0			select 2		WRITE					
ステート1										
ステート2				select 1		WRITE				
ステート3			select 1	select 2	WRITE					
ステート4	active			select 1		WRITE	WRITE	WRITE		
ステート5			select 1	select 2	WRITE				WRITE	WRITE
ステート6	active	active		select 1		WRITE	WRITE	WRITE		
ステート7			select 1	select 2	WRITE				WRITE	WRITE
ステート8	active	active					WRITE	WRITE		
ステート9									WRITE	WRITE
ステート10		active								
ステート11										

【0103】　　＊　＊【表13】

現状態	次状態	
ステート0	ステート1	初期値設定
ステート1	ステート2（S1＝1） END　（S1＝0）	初期値判断
ステート2	ステート3	パイプライン導入部
ステート3	ステート4（S1＝1） ステート8（S1＝0）	
ステート4	ステート5	
ステート5	ステート6（S1＝1） ステート8（S1＝0）	
ステート6	ステート7	フル・パイプライン
ステート7	ステート8（S1＝1） ステート8（S1＝0）	
ステート8	ステート9	パイプライン導出部
ステート9	ステート10	
ステート10	ステート11	
ステート11	END	

【0104】表12及び表13並びに図11に、表10のリソースバインディングと表11のスケジューリングに対して、リソース接続と制御論理の生成を行ない、パイプラインアーキテクチャ合成した結果を示す。図11において、セレクタ28が新たにつけ加えられたセレク※タであり、それによって、加算器23を共有化している。

【0105】実施例5．次に、この発明の第5実施例の高位合成方法について説明する。第1実施例の高位合成方法では、演算器間に制御ステップの差異分の段数のパ

イプラインレジスタが生成される。しかし、差異が大きい場合、パイプラインレジスタを用いてデータを保持するより、必要な時点で再計算した方が良い場合がある。このような場合に、データパス構造の構成に変更を加える工程を追加する。

【０１０６】図８のデータパス構造には、Adr，Adr（１），Adr（２）と続くパイプラインレジスタが存在する。この例では、３サイクル前のｉ＋baseの演算結果の値を保持するために、パイプラインレジスタが作られている。しかし、ｉは１サイクルに１インクリメントされているので、３サイクル前のｉ＋baseの値は、現在の加算器の出力から、３を減じることによって求めることもできる。もし、２段のパイプラインレジスタ（１段目のパイプラインレジスタは、消去できない。）を設けるよりも、１つの演算器を導入して値を再計算した方がＬＳＩの面積（コスト）が小さくなる場合、パイプラインレジスタを設けずに減算器を新設する。コンピュータによってコスト計算を行いながら高位合成を行うことは、従来から実施されている。

【０１０７】図１２に、図８のデータパス構造のパイプラインレジスタを再計算モジュールで置換した例を示す。図１２において、２９が減算器である。減算器２９によって、レジスタｒ６に記憶されている値からレジスタｒ１３に記憶されている値を引くことによって、出力装置２７に対して、レジスタｒ６が、３ステップ前に保持していた値を与えることができる。

【０１０８】なお、ここでは、ステップ毎に１ずつ増加する値を保持する場合について示したが、ステップ毎に値が減少する場合であってもよく、また、他の規則によって値が変わる場合であっても同様の効果を奏する。

【０１０９】次に、この発明の高位合成方法の全体の流れについて説明する。高位合成方法は、ハードウェア記述言語を用いた、動作レベルの記述を入力し、論理合成ツールが直接読み込み可能なレジスタトランスファレベルのハードウェア記述言語による記述を出力する。図１３は、この発明の高位合成方法の手順を示すフローチャートである。

【０１１０】工程Ｓ３１において、動作レベルのハードウェア記述言語で表現された動作記述を入力し、オペレーション系列へと変換する。工程Ｓ３２において、オペレーション系列からＣＤＧを生成する。工程Ｓ３３において、第１実施例と同様に、パイプライン状の実行が必要な記述部に対して、パイプライン状の実行が可能な形態へと、ＣＤＧを変形する。

【０１１１】工程Ｓ３４において、変形されたＣＤＧに基づき、第１実施例ないし第５実施例の工程を適用して、パイプラインアーキテクチャを合成する。工程Ｓ３５において、パイプライン状の実行が不要な記述部に対して、従来方式にて、アーキテクチャを合成する。この時、工程Ｓ３４で得られたアーキテクチャ中の演算器を共有してもよい。最後に、工程Ｓ３６において、合成されたアーキテクチャを、論理合成ツールが直接読み込むことのできる、レジスタ・トランスファ・レベルのハードウェア記述言語による記述として出力する。

【０１１２】実施例６．次に、この発明の第６実施例による高位合成方法について説明する。ループ記述を、コントロールフローグラフと、データ依存グラフに変換する時、通常は、一つの演算子に対して、データ依存グラフ上の１つのノードが生成される。しかし、複雑な演算を実行するゲート回路をユーザが設計し、それに対応する演算子をユーザが追加したい場合がありうる。第６実施例の高位合成方法は、これを容易に可能とする手法に関わるものである。

【０１１３】ループ動作を含むハードウェアの動作を記述するのに適当な言語として、ＶＨＤＬ，Verilog －ＨＤＬ等のハードウェア記述言語が考えられる。これらの言語には、複数の入力データを与え、１つの出力を返す関数（function）文がある。

【０１１４】通常は、この関数文の内部動作まで解析し、演算子レベルにまで分解してＣＤＧを生成する。図１４は、Verilog －ＨＤＬ記述からデータ依存グラフへの通常の変換を示す図である。Verilog －ＨＤＬ記述４０は、データ依存グラフ５０に変換される際、一塊の関数５０ａも演算子のレベルで展開して作成されている。

【０１１５】しかし、変換ツールへの特殊な指示子をコメント文として挿入するようにすることで、この関数を新しいユーザ定義演算子とみなすようにする。

【０１１６】これにより、容易に新しいユーザ定義演算子を増やすことができる。図１５に、Verilog-ＨＤＬでのユーザ定義演算子の例を示す。まず、ｆｕｎｃ６０というユーザ定義演算子をコンピュータにあらかじめ登録しておく。そして、Verilog－ＨＤＬ記述４５を作成する際に、”／＊ＵｓｅｒＤｅｆｉｎｅｄＦｕｎｃｔｉｏｎ＊／”という、ユーザ定義演算子の使用を宣言する文を挿入する。そうすることで、データ依存グラフ作成の際も、ｆｕｎｃという関数を一つのノードに割り付けることができる。ｆｕｎｃという関数に対応する回路は、予めコンピュータに登録されているため、設計者が意図した結果が得られ、性能の良いＬＳＩの設計が可能になる。

【０１１７】実施例７．次に、この発明の第７実施例による高位合成方法について説明する。第７実施例の高位合成方法は、入力記述を自動修正し、ループ回数を削減することで、処理速度とリソース数のトレードオフを達成する手法に関するものである。

【０１１８】合成されたパイプラインアーキテクチャは、ループ部の動作にＮ＋α（Ｎはループ回数、αは定数）サイクル必要とする。従って処理の高速化を図るためには、ループ回数を削減すればよい。

【０１１９】ループ回数を削減する単純な方法は、数回

分のループを1回分に展開することである．例えば、第1実施例の高位合成方法において、2回分のループの記述を1回分に展開する工程をさらに追加することで、ループは、N／2＋β（Nはループ回数、βは定数）サイクルとなり、処理速度は、約2倍となる．数3に展開前の記述を、数4に展開後の記述を示す．

【0120】

【数3】

```
for(i=0;i<N;i=i+1)begin
  Adr=i+base;       /*set address*/
  Data=input(Adr); /*input from a port*/
  Res=Data*Coef;    /*calculation*/
  output(Adr,Res); /*output to a port*/
end
```

【0121】

【数4】

```
for(i=0;i<N;i=i+2)begin
  i_tmp=i;
  Adr=i_tmp+base;/*set address*/
  Data=input(Adr); /*input from a port*/
  Res=Data*Coef; /*calculation*/
  output(Adr,Res); /*output to a port*/

i_tmp=i+1;
  Adr=i_tmp+base;/*set address*/
  Data=input(Adr); /*input from a port*/
  Res=Data*Coef; /*calculation*/
  output(Adr,Res); /*output to a port*/
end
```

【0122】なお、上記各実施例は、それぞれ組み合わせて用いることもでき、その場合にも、それぞれの固有の効果を失うことはない。

【0123】

【発明の効果】以上のように、請求項1記載の発明の自動高位合成方法によれば、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフへと変形する変形工程を備えているので、新たなコントロールフローグラフ及びデータ依存グラフを用いて、並列度の高いパイプラインアーキテクチャを自動的に合成できるという効果がある．

【0124】請求項2記載の発明の自動高位合成方法によれば、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフへと変形する変形工程を容易に実現することができるという効果がある．

【0125】請求項3記載の発明の自動高位合成方法によれば、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフを用いて、演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器の制御理論及びデータパス構造を容易に得ることができるという効果がある．

【0126】請求項4記載の発明の自動高位合成方法によれば、計算機において、演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための前記演算器の制御論理及びデータパス構造を自動生成する生成工程を容易に実現できるという効果がある．

【0127】請求項5記載の発明の自動高位合成方法によれば、演算器をラッチで分離されたパイプライン演算器としてモデル化することによって、1クロック周期以上の遅延を有する演算器をパイプラインアーキテクチャにおいて使用できるようにすることができるという効果がある．

【0128】請求項6記載の発明の自動高位合成方法によれば、パイプラインピッチを増やすことによって、1クロック周期以上の遅延を有する演算器をパイプラインアーキテクチャにおいて使用できるようにすることができるという効果がある．

【0129】請求項7記載の発明の自動高位合成方法によれば、一部の演算器を共有にすることによって、ＬＳＩの規模を縮小することがきてるという効果がある．

【0130】請求項8記載の発明の自動高位合成方法によれば、パイプラインレジスタに換えて所定の演算を行う回路を用いて、ＬＳＩのサイズを縮小できるという効果がある．

【0131】請求項9記載の発明の自動高位合成方法によれば、入力を動作レベルのハードウェア記述言語で行って、レジスタトランスファレベルのハードウェア記述言語で記述されたパイプラインアーキテクチャを得ることができるという効果がある．

【0132】請求項10記載の発明の自動高位合成方法によれば、ユーザが新たに演算器を登録することができ、より性能の良いパイプラインアーキテクチャを構成することができるという効果がある．

【0133】請求項11記載の発明の自動高位合成方法によれば、展開したループを用いてパイプラインアーキテクチャを生成することができ、ループの回数を減らして処理速度の速いＬＳＩを得ることができるという効果がある．

【図面の簡単な説明】

【図1】　ループの実行状態を示す概念図である．

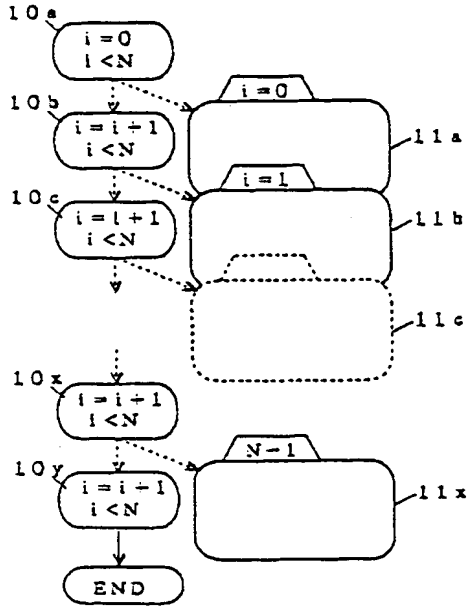
【図2】　ループの非依存部の動作をパイプライン状に実行する各ノードの動作を示す概念図である．

【図3】　コントロールフローグラフとデータ依存グラフの変形を行なう手順を示すフローチャートである．

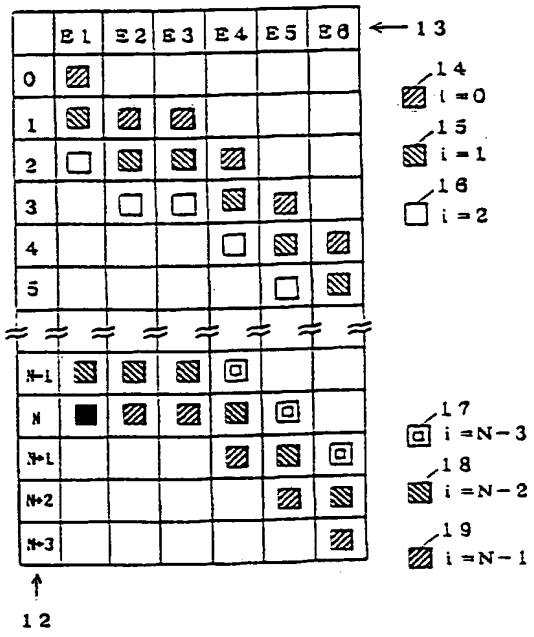
【図4】　コントロールフローグラフとデータ依存グラフの変形を行なう手順を示すフローチャートである．

【図5】　パイプライン状の実行が可能な形態のコントロールフローグラフとデータ依存グラフである。
【図6】　パイプラインアーキテクチャを合成する手続のフローチャートである。
【図7】　パイプラインアーキテクチャを合成する手続のフローチャートである。
【図8】　パイプラインアーキテクチャ合成用のリソース接続結果を示すブロック図である。
【図9】　1クロック周期以上の遅延時間をもつ演算器を示す概念図である。
【図10】　1クロック周期以上の遅延時間をもつパイプライン演算器を示す概念図である。
【図11】　演算器を共有化した場合のパイプラインアーキテクチャを示すブロック図である。
【図12】　パイプラインレジスタの再計算モジュールでの置換を示すブロック図である。
【図13】　パイプラインアーキテクチャを合成する高位合成方法全体の流れを示すフローチャートである。
【図14】　関数を構成する演算子の記述の変換を説明するための図である。
【図15】　ユーザ定義演算子の記述の変換を説明するための図である。
【図16】　従来の高位合成方法の手順を示すフローチャートである。
【図17】　数1のループ記述に対応したコントロールフローグラフとデータ依存グラフである。
【図18】　従来の高位合成方法により合成されるデータパス構造を示すブロック図である。
【図19】　従来の高位合成方法により合成されたアーキテクチャにおける演算器の動作状況を示す図である。
【図20】　従来の高位合成方法により合成されたコントロールグラフとデータ依存グラフである。
【符号の説明】
1～5　ブロック、r1～r12　レジスタ、20，28　セレクタ、21比較器、22　制御論理回路、23，24　加算器、25　入力装置、26　乗算器、27　出力装置。

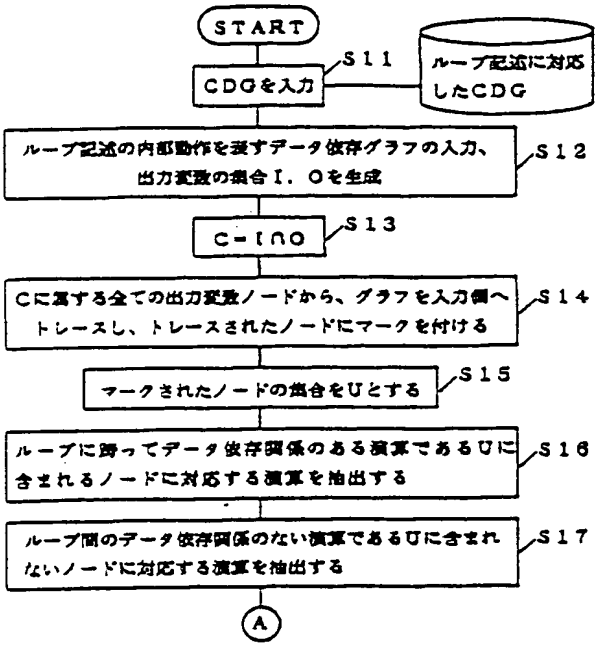
【図1】



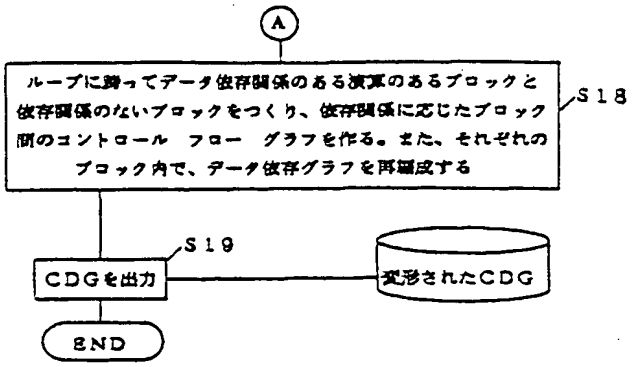
【図2】



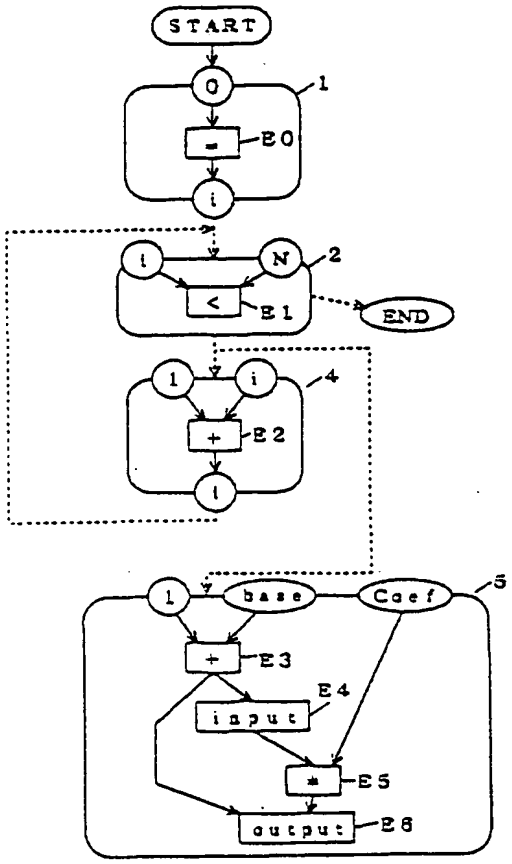
【図3】


START
CDGを入力
S11
ループ記述に対応したCDG
ループ記述の内部動作を表すデータ依存グラフの入力、出力変数の集合Ｉ．Ｏを生成
S12
C=ino
S13
Cに属する全ての出力変数ノードから、グラフを入力側へトレースし、トレースされたノードにマークを付ける
S14
マークされたノードの集合をUとする
S15
ループに跨ってデータ依存関係のある演算であるUに含まれるノードに対応する演算を抽出する
S16
ループ間のデータ依存関係のない演算であるUに含まれないノードに対応する演算を抽出する
S17
A

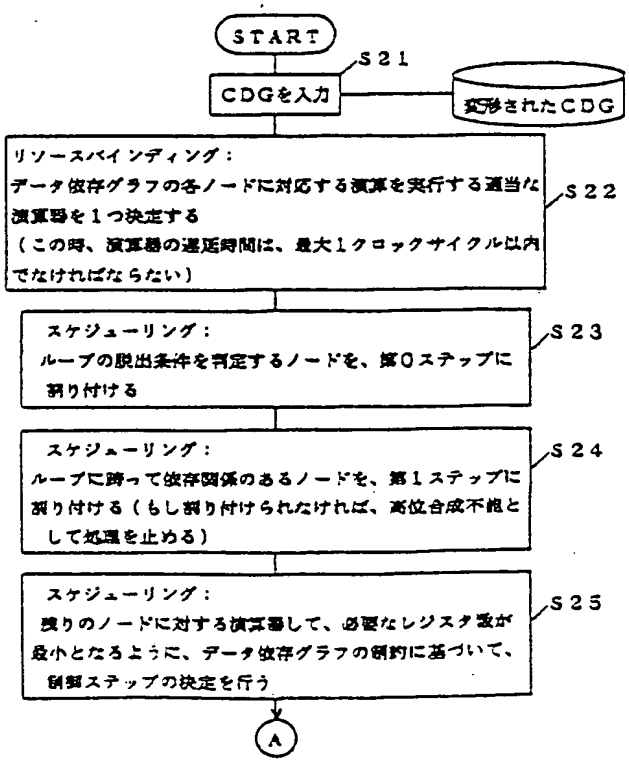
【図4】


A
ループに跨ってデータ依存関係のある演算のあるブロックと依存関係のないブロックをつくり、依存関係に応じたブロック間のコントロール　フロー　グラフを作る。また、それぞれのブロック内で、データ依存グラフを再編成する
S18
CDGを出力
S19
変形されたCDG
END

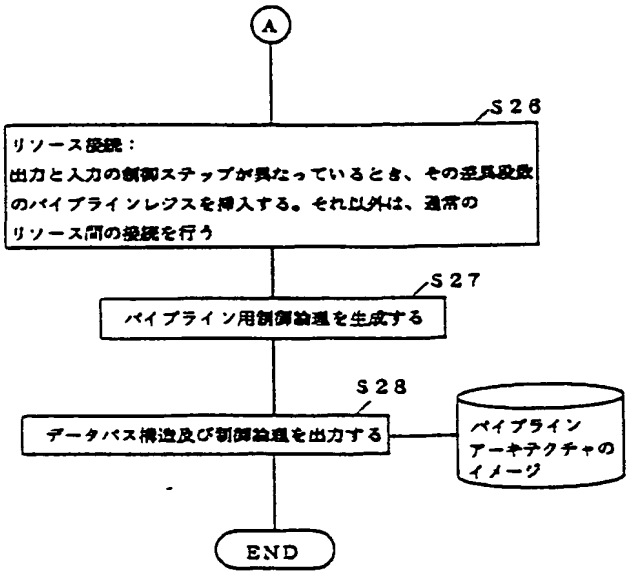
【図5】


START
0
1
=
E0
i
i
N
2
<
E1
END
1
i
4
+
E2
1
1
base
Coef
5
+
E3
E4
input
*
E5
output
E6

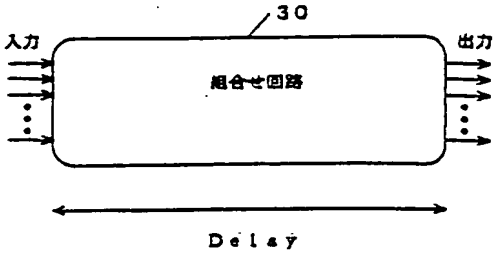
【図6】


START
CDGを入力
S21
変形されたCDG
リソースバインディング：
データ依存グラフの各ノードに対応する演算を実行する適当な演算器を１つ決定する
（この時、演算器の遅延時間は、最大１クロックサイクル以内でなければならない）
S22
スケジューリング：
ループの脱出条件を判定するノードを、第０ステップに割り付ける
S23
スケジューリング：
ループに跨って依存関係のあるノードを、第１ステップに割り付ける（もし割り付けられなければ、高位合成不能として処理を止める）
S24
スケジューリング：
残りのノードに対する演算器して、必要なレジスタ数が最小となるように、データ依存グラフの制約に基づいて、制御ステップの決定を行う
S25
A

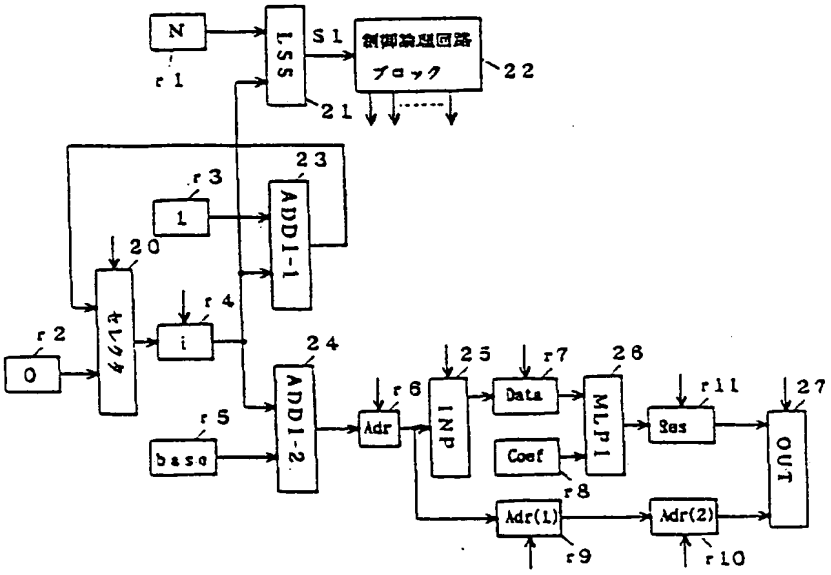
【図7】


A
S26
リソース接続：
出力と入力の制御ステップが異なっているとき、その差異段数のパイプラインレジスタを挿入する。それ以外は、通常のリソース間の接続を行う
S27
パイプライン用制御論理を生成する
S28
データパス構造及び制御論理を出力する
パイプラインアーキテクチャのイメージ
END

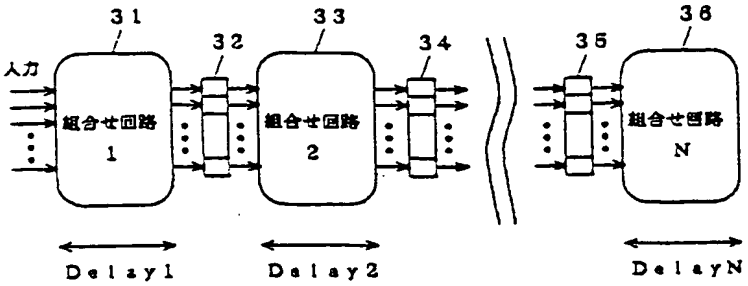
【図9】


30
入力
出力
組合せ回路
Delay

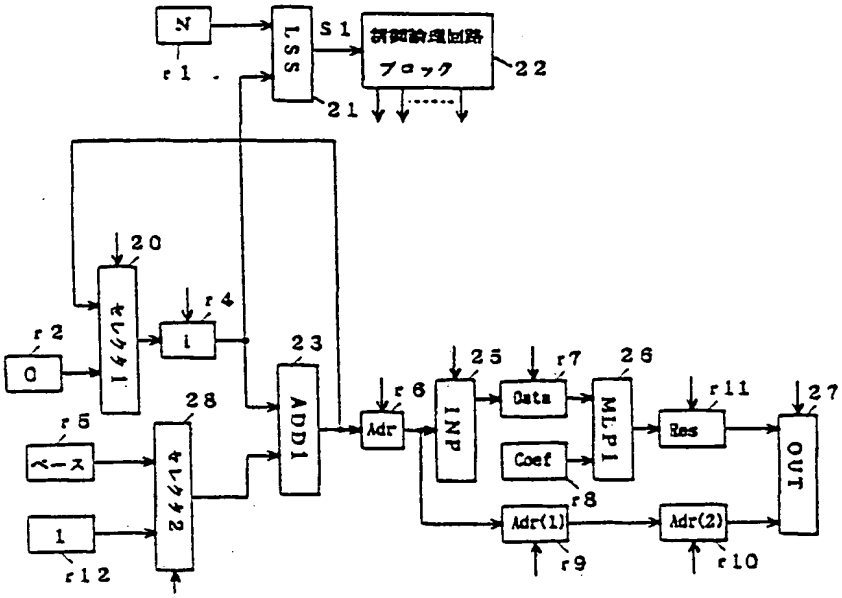
【図8】


N
r1
LSS
S1
制御論理回路ブロック
22
21
23
r3
1
ADD1-1
20
r4
i
r2
0
セレクタ
24
ADD1-2
r5
base
r6
Adr
25
INP
r7
Data
Coef
r8
Adr(1)
r9
26
MLPY1
r11
Res
27
OUT
Adr(2)
r10

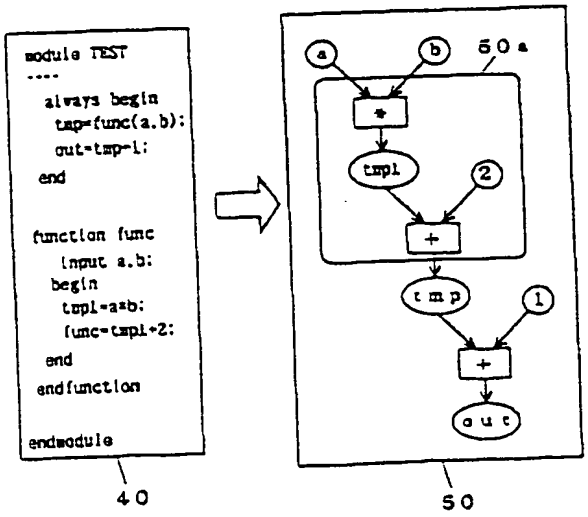
【図10】



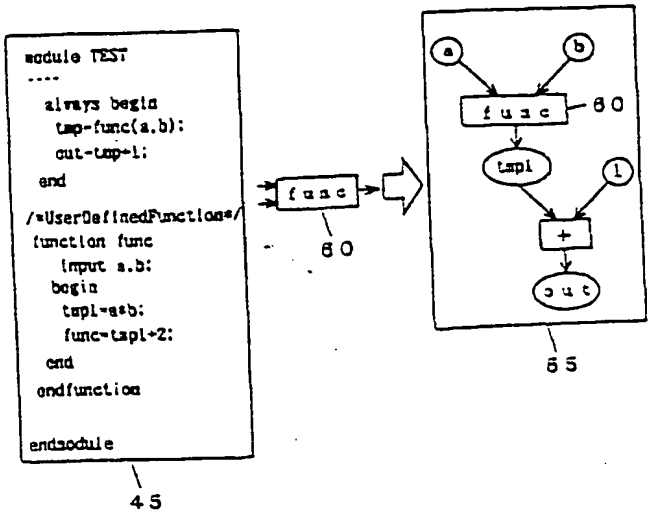
【図11】



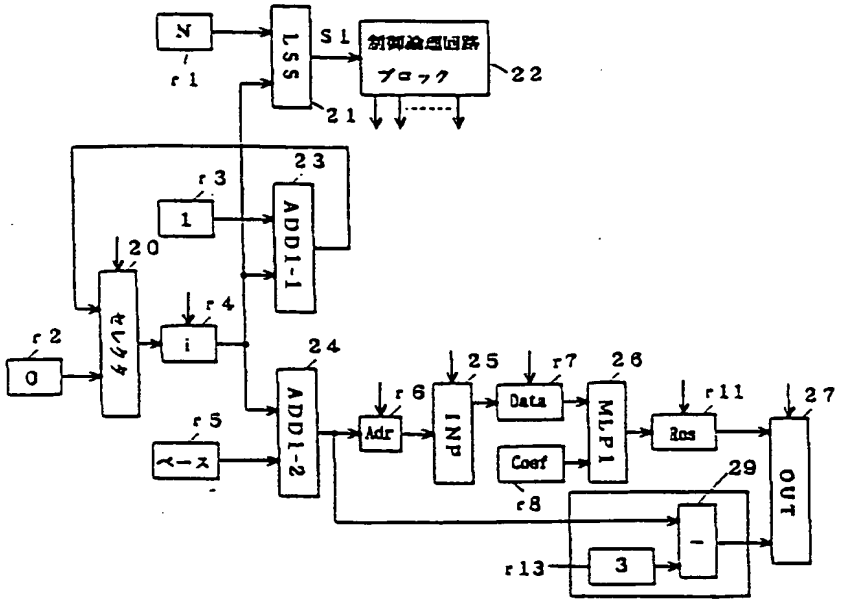
【図14】



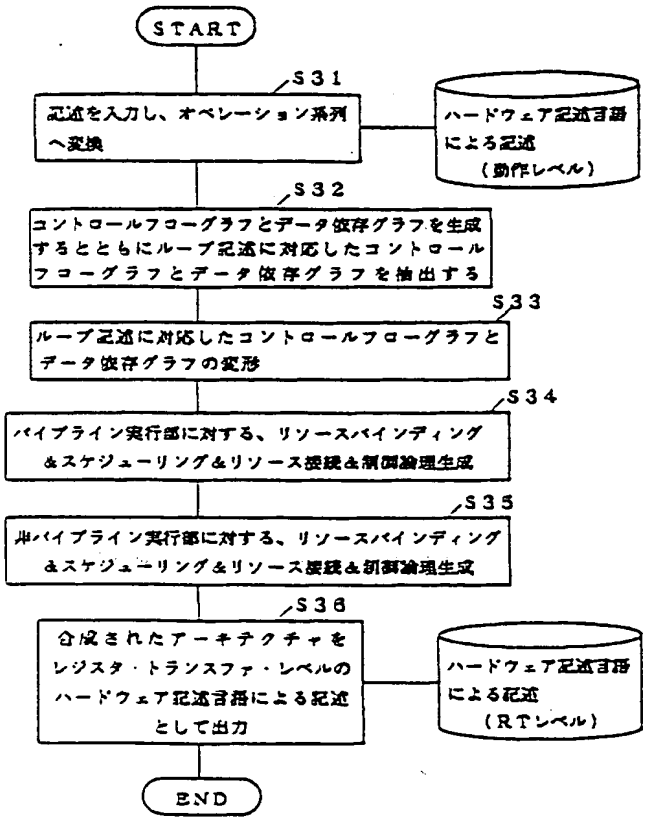
【図15】



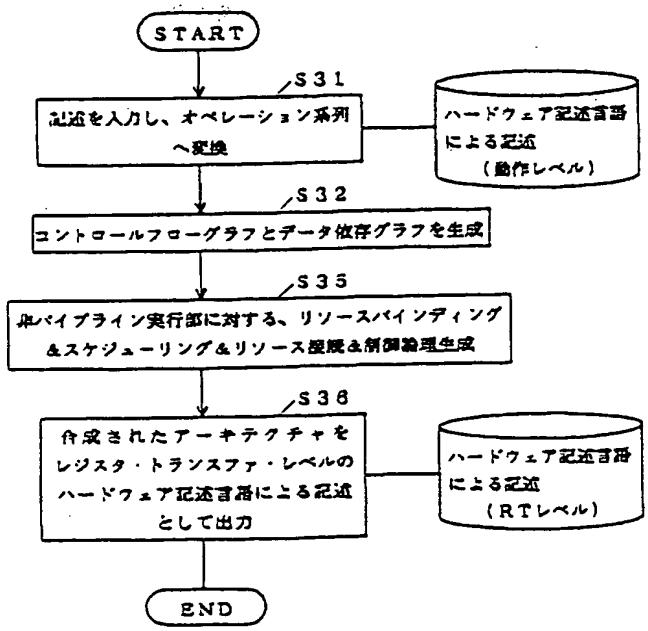
【図12】


N
r1
LSS
S1
制御論理回路
ブロック
22
21
23
r3
1
ADD1-1
20
レジスタ
r4
i
r2
0
24
ADD1-2
r5
ベース
r6
Adr
25
IND
r7
Data
Coef
r8
26
MLPL1
r11
Res
27
OUT
29
−
r13
3

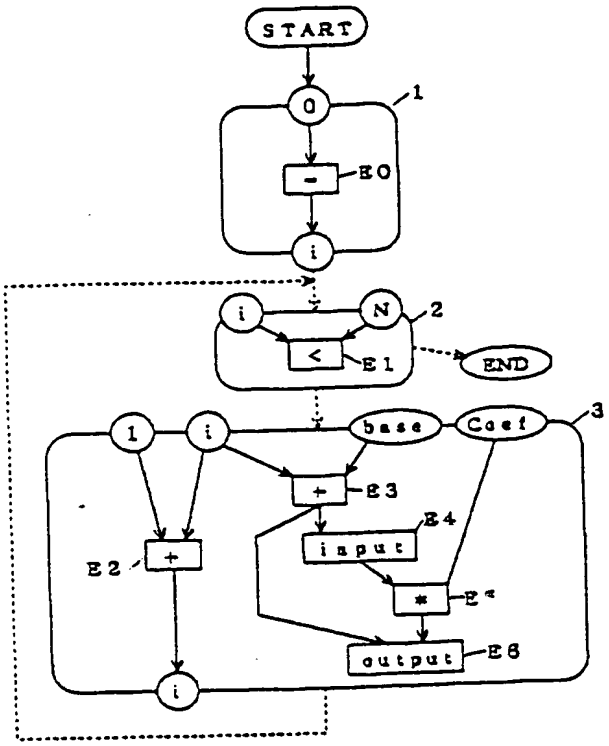
【図13】


START
S31
記述を入力し、オペレーション系列へ変換
ハードウェア記述言語による記述（動作レベル）
S32
コントロールフローグラフとデータ依存グラフを生成するとともにループ記述に対応したコントロールフローグラフとデータ依存グラフを抽出する
S33
ループ記述に対応したコントロールフローグラフとデータ依存グラフの変形
S34
パイプライン実行部に対する、リソースバインディング＆スケジューリング＆リソース接続＆制御論理生成
S35
非パイプライン実行部に対する、リソースバインディング＆スケジューリング＆リソース接続＆制御論理生成
S36
合成されたアーキテクチャをレジスタ・トランスファ・レベルのハードウェア記述言語による記述として出力
ハードウェア記述言語による記述（RTレベル）
END

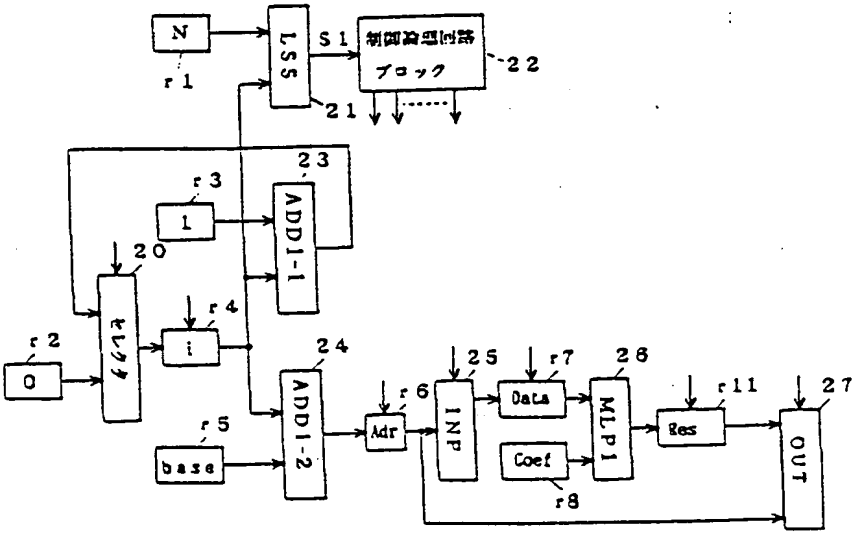
【図16】


START
S31
記述を入力し、オペレーション系列へ変換
ハードウェア記述言語による記述（動作レベル）
S32
コントロールフローグラフとデータ依存グラフを生成
S35
非パイプライン実行部に対する、リソースバインディング＆スケジューリング＆リソース接続＆制御論理生成
S36
合成されたアーキテクチャをレジスタ・トランスファ・レベルのハードウェア記述言語による記述として出力
ハードウェア記述言語による記述（RTレベル）
END

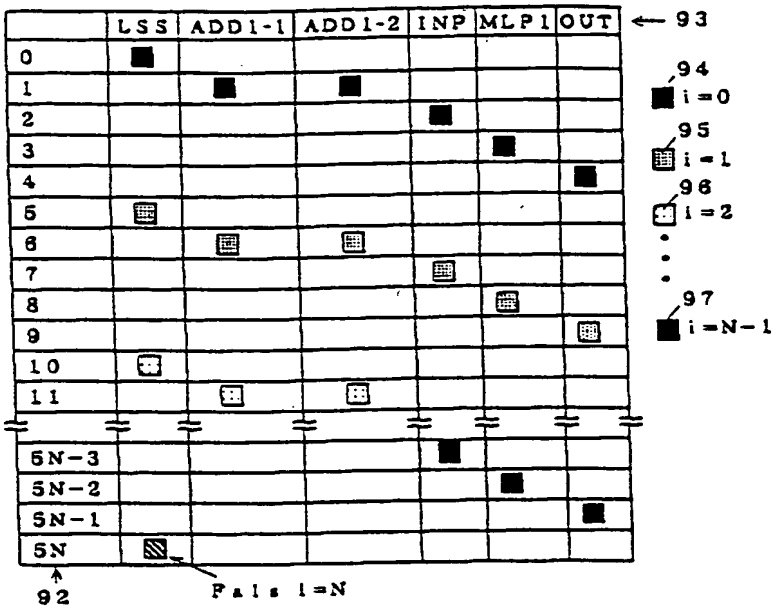
【図17】


START
0
1
－
E0
i
i
N
2
<
E1
END
1
i
base
Coef
3
+
E3
input
E4
E2
+
*
E5
output
E6
i

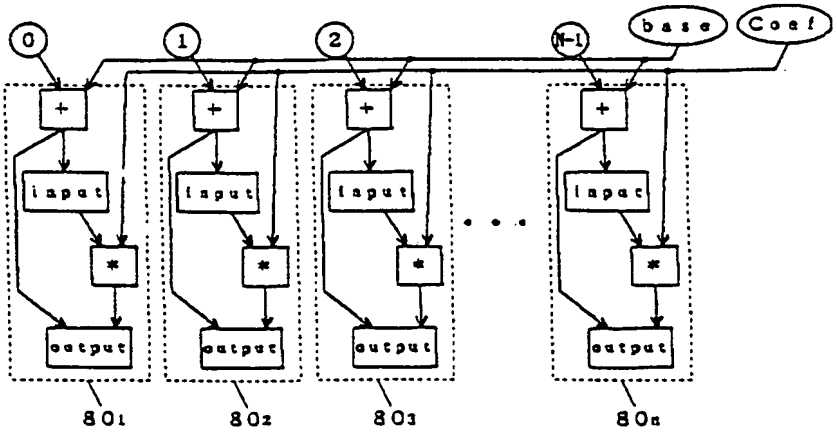
【図18】


N
r1
LSS
S1
制御論理回路
ブロック
22
21
23
r3
1
ADD1-1
20
r4
r2
0
i
24
ADD1-2
r5
base
r6
Adr
25
r7
26
r11
27
INP
Data
MLP1
Res
OUT
Coef
r8

【図19】


LSS
ADD1-1
ADD1-2
INP
MLP1
OUT
93
0
1
2
3
4
5
6
7
8
9
10
11
5N-3
5N-2
5N-1
5N
92
Pals i=N
94
i=0
95
i=1
96
i=2
97
i=N-1

【図20】


0
1
2
N-1
base
Coef
+
input
*
output
801
802
803
80n